Lecture

10

EEM216A
Fall 2012

Timing Analysis

Prof. Dejan Marković
ee216a@gmail.com

Two Types of Machines with State

And two quite different abstract models:

- **Data storage used for computation (Data Flows)**
- **States for sequencing information (Finite State Machines)**

Data Flows (Storage for Computation)

- **The storage holds data that is being manipulated. The (enormous) number of bits does not matter. It is simply the data-set that is being manipulated.**
- **State is not that important, it is the flow of data that is critical**

10.3

FSMs (States for Sequencing of Information)

- **The storage is used to hold your place in some decision-making process. It indicates where you are, and using this information you decide what to do next.**
- **The amount of state (number of unique decision points) is finite, and usually limited**
  - One could draw out the "decision graph" showing the possible transitions between states

10.4

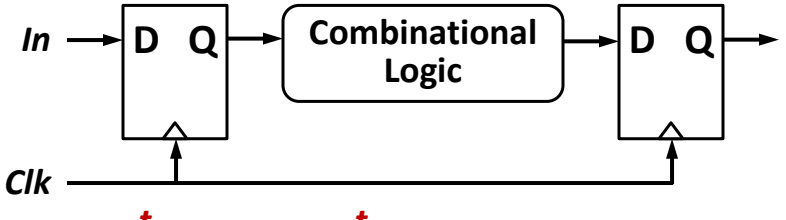
Timing Analysis

Timing constraints:

- **Long path: Setup time**
  - Leads to cycle time violation
  - **Fix:** increase cycle time
    (can be done during chip operation)
- **Short path: Hold time**
  - Leads to functional violation
  - **Fix:** insert buffers
    (can only be done at design time)

- **Clock nonidealities (skew and jitter) directly impact timing constraints**

Timing: Cycle Time & Race Margin



$t_{Clk\text{-}Q}$
t_{Setup}
$t_{Clk\text{-}Q,cd}$
t_{Hold}

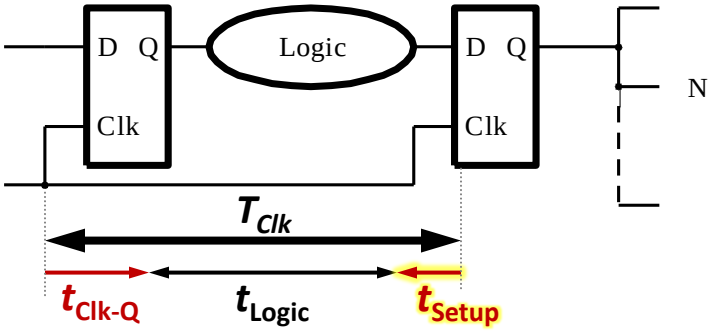
t_{Logic}
$t_{Logic,cd}$

Cycle time : $T_{Clk} > t_{Clk\text{-}Q} + t_{Logic} + t_{Setup}$

Race margin : $t_{Hold} < t_{Clk\text{-}q,cd} + t_{Logic,cd}$

The Setup / Cycle Time Constraint

Consecutive Clk edges

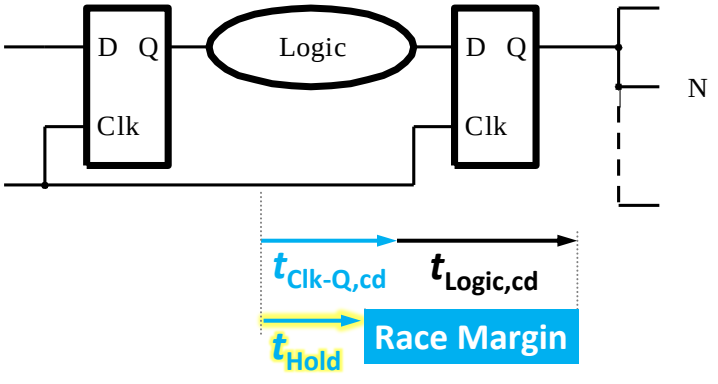


$$T_{Clk} > t_{Clk-Q} + t_{Logic} + t_{Setup}$$

10.7

The Hold / Race Margin Constraint

Concurrent Clk edges

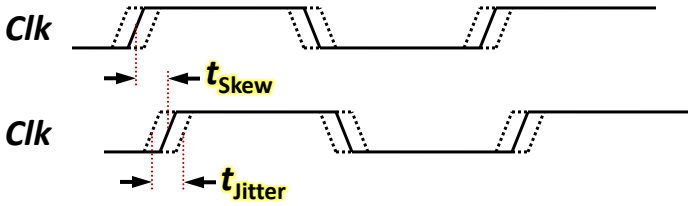


$$t_{Hold} < t_{Clk-Q,cd} + t_{Logic,cd}$$

10.8

Clock Nonidealities

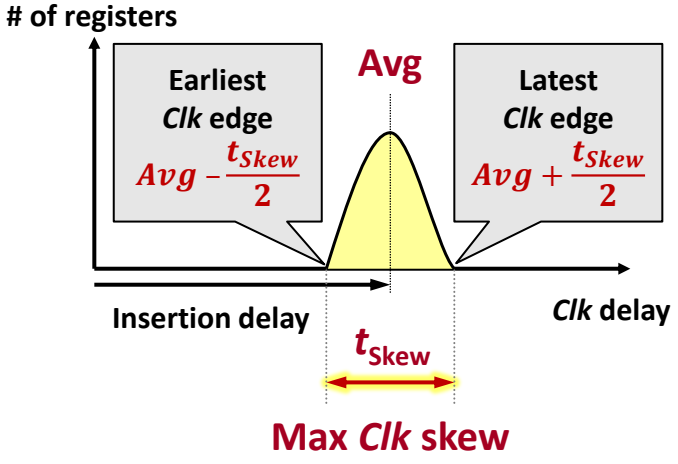
- **Skew: spatial variation in temporally equivalent clock edges; deterministic + random, t_{Skew}**



- **Clock jitter: temporal variations in consecutive edges of the clock signal; modulation + random noise, t_{Jitter}**
- **Variation of the pulse width**
  - For level-sensitive clocking

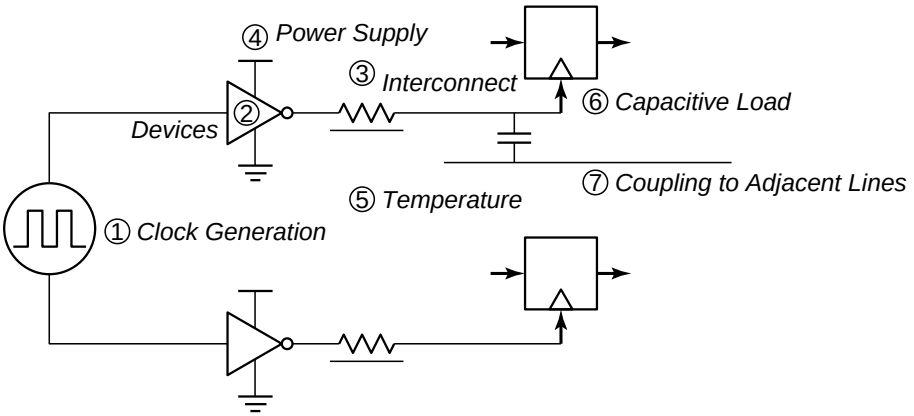
Clock Skew

Distribution of clock tree insertion delay



Max *Clk* skew

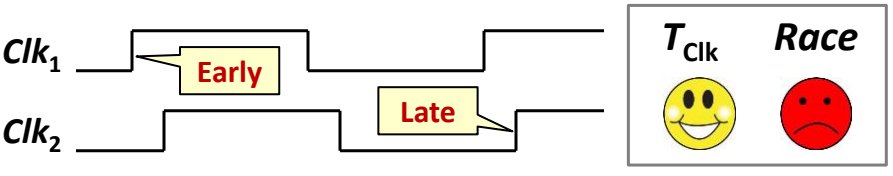
Sources of Skew and Jitter



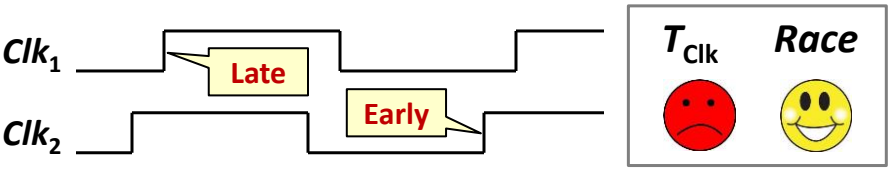
10.11

Positive and Negative Skew

Positive: early – late case



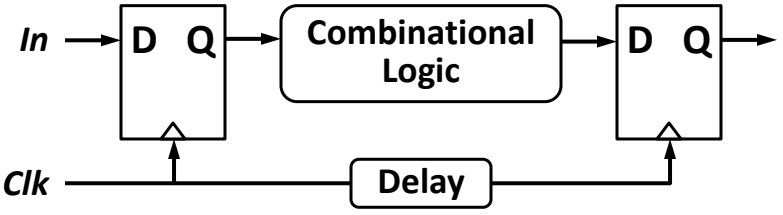
Negative: late – early case



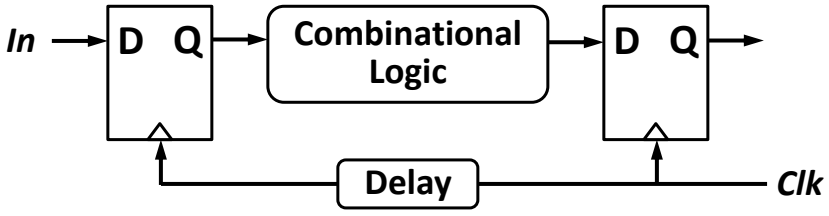
10.12

Signal Routing for Positive and Negative Skew

Positive skew: *Clk* and *Data* routed in the same direction



Negative skew: *Clk* and *Data* routed in opposite directions



10.13

Skew + Jitter = Clock Uncertainty

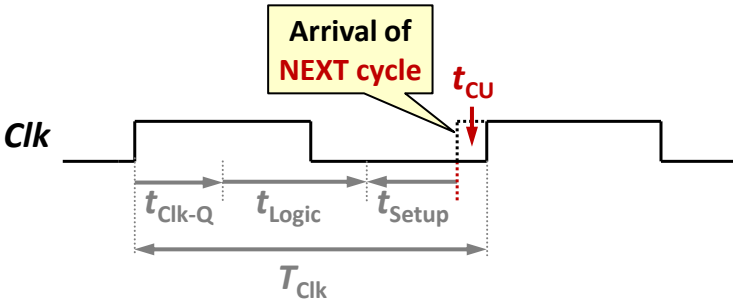
Single parameter used in CAD tools

Clock uncertainty, t_{CU}

10.14

Impact of t_{CU} on Timing: Cycle Time

Cycle time (long path): late – early analysis

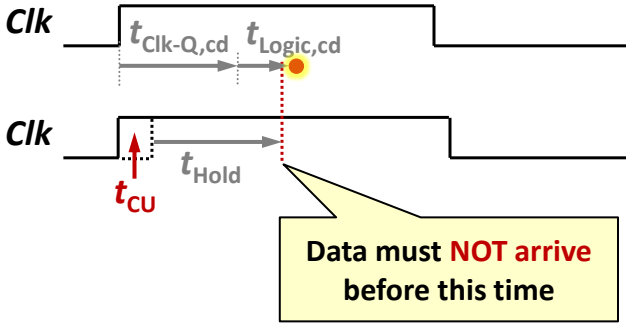


$$T_{Clk} > t_{Clk-Q} + t_{Logic} + t_{Setup} + |t_{CU}|$$

10.15

Impact of t_{CU} on Timing: Race Margin

Race immunity (short path): early – late analysis



$$t_{Hold} + |t_{CU}| < t_{Clk-Q,cd} + t_{Logic,cd}$$

10.16

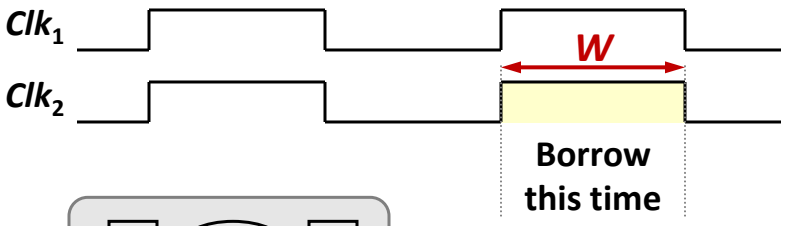
Time Borrowing

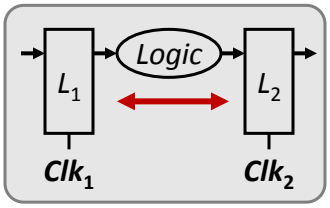
Time Borrowing: Classification

- **Dynamic: scheduling data to arrive to transparent CSE**
  - No "hard" boundaries between stages
  - In latch-based level sensitive or soft-edge clocking
- **Static: control delay between clock inputs**
  - Clocks scheduled to arrive so that the slower paths obtain more time to evaluate, taking away the time from faster paths
  - It can operate with conventional hard-edge CSEs
  - Also called *opportunistic skew scheduling*

Dynamic Time Borrowing

- **Latch-based designs:** clock pulse-width can be borrowed if the next stage can pay it back (with faster logic)

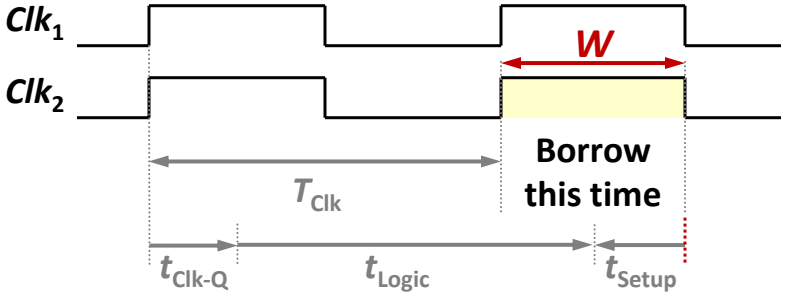




- **Cycle time?**
- **Race margin?**

Cycle Time Analysis

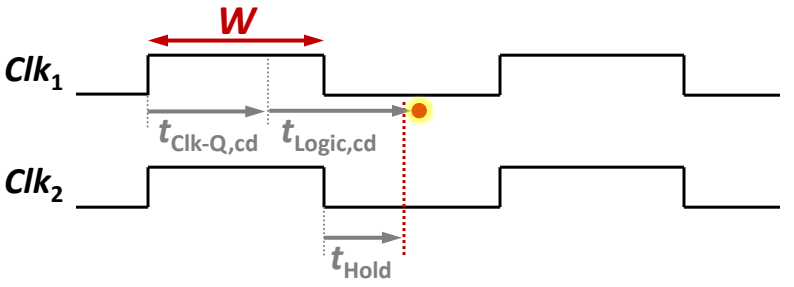
Logic delay can be extended by W



$$T_{Clk} + W > t_{Clk\text{-}Q} + t_{Logic} + t_{Setup}$$

Race Margin Analysis: No Time Borrowed

$t_{Clk\text{-}Q}$ matters in this case

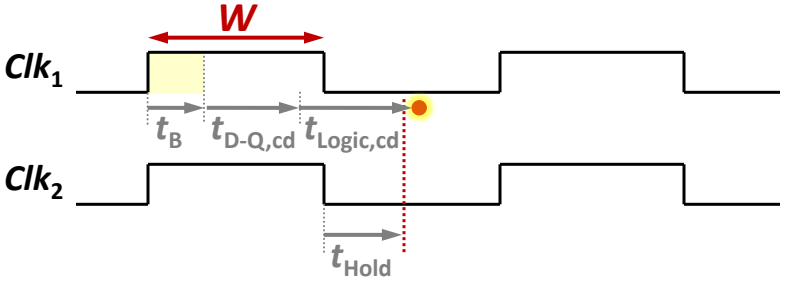


$$t_{Hold} + W < t_{Clk\text{-}Q,cd} + t_{Logic,cd}$$

10.21

Race Margin Analysis: Borrowed Time

Account for borrowed time, t_B



$$t_{Hold} + W < t_B + t_{D\text{-}Q,cd} + t_{Logic,cd}$$

10.22

Race Margin Analysis: Summary

No time borrowed (more likely critical case)

$$t_{Hold} + W < t_{Clk\text{-}Q,cd} + t_{Logic,cd}$$

Borrowed time, t_B

$$t_{Hold} + W < t_B + t_{D\text{-}Q,cd} + t_{Logic,cd}$$

Min{$t_B + t_{D\text{-}Q,cd}$, $t_{Clk\text{-}Q,cd}$} critical

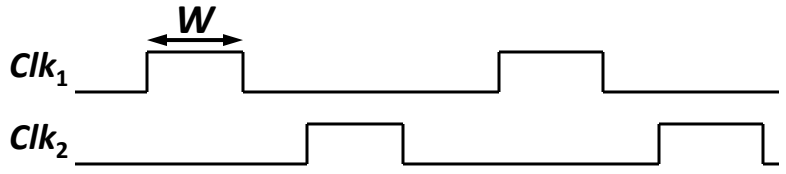
10.23

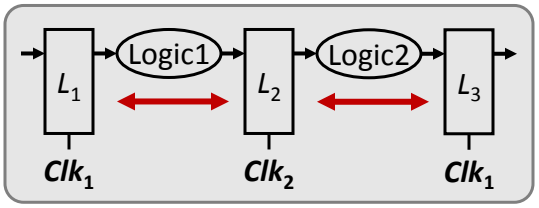
Two-Phase Clocking

10.24

Two-Phase Clocking

Freedom to control two phases *and* pulse-width

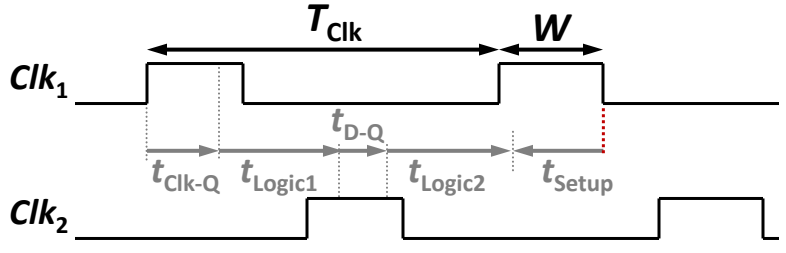




- **Cycle time?**
- **Bounds on *W*?**
- **Max t_{CU}?**

Two-Phase Clocking: Cycle Time

Assume $t_{Clk\text{-}Q} + t_{Setup} = t_{D\text{-}Q}$



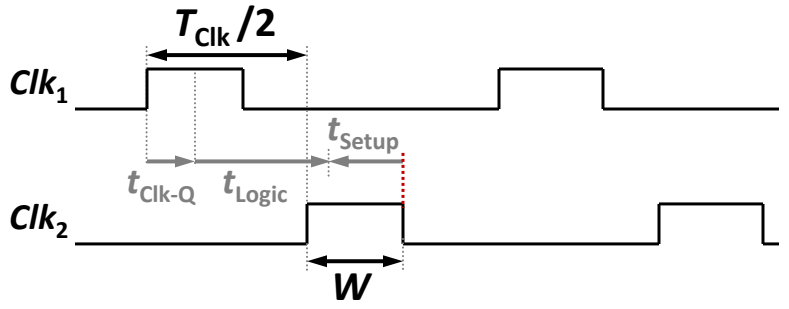
$$T_{Clk} + W > t_{Clk\text{-}Q} + t_{Logic1} + t_{D\text{-}Q} + t_{Logic2} + t_{Setup}$$

$$T_{Clk} + W > t_{Logic1} + t_{Logic2} + 2t_{D\text{-}Q}$$

Two-Phase Clocking: Min *W*

Max delay

$$T_{Clk}/2 + W > t_{Logic,max} + t_{D\text{-}Q}$$

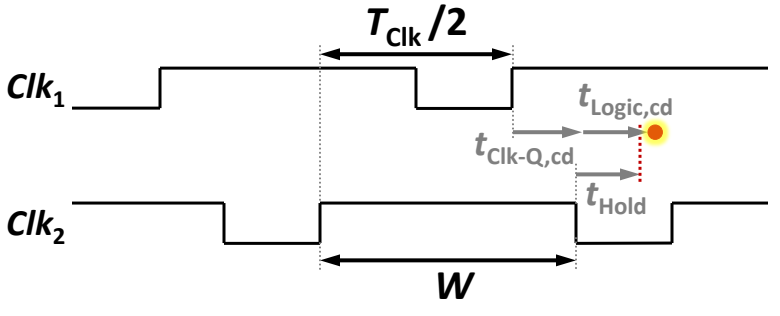


10.27

Two-Phase Clocking: Max *W*

Min delay

$$t_{Hold} + W < T_{Clk}/2 + t_{Clk\text{-}Q,cd} + t_{Logic,cd}$$



10.28

Two-Phase Clocking: Max t_{CU}

***W* guards against t_{CU}**

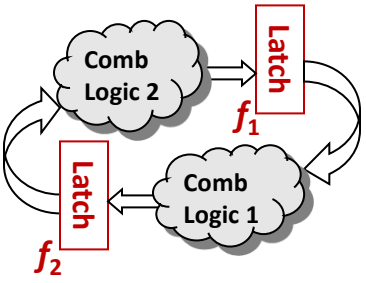
$$t_{CU} < W_{max} - W_{min}$$

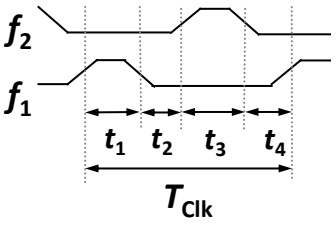
- **Assuming $t_{CU} = 2|t_{Skew}|$**
  - Both polarities of t_{Skew}

$$|t_{Skew}| < (W_{max} - W_{min})/2$$

10.29

Summary: 2-Φ Clocking



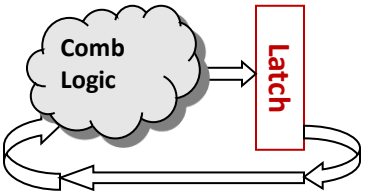


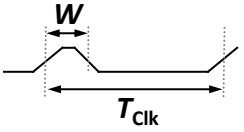
- **Max($t_{CL1/2}$) < $t_1 + t_{2/4} + t_3 - t_{Setup} - t_{Clk\text{-}Q} - t_{CU}$**
- **Strictly, Max($t_{CL1} + t_{CL2}$) < $T_{Clk} - 2t_{D\text{-}Q}$**
- **Min($t_{CL1/2}$) > $t_{1/3} + t_{Hold} - t_{Clk\text{-}Q} + t_{CU}$**

- **More clocking overhead (2 clocks) and low f_{max}**

10.30

Clocking Methodology: Pulse-Mode

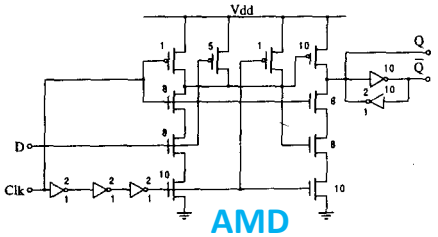




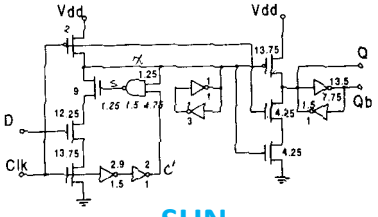
$$t_{CL} < T_{Clk} - t_{D\text{-}Q}$$

$$t_{CL} > W + t_{Hold} - t_{Clk\text{-}Q} + t_{CU}$$

Examples



AMD



SUN

10.31

Do We Need Clocks?

10.32

Minimum Clock Cycle Time Revisited

- **Cycle time determined by the delay through logic**
  - It must arrive before the latching edge
  - If too late, it waits until the next cycle
    - Synchronization and sequential order is off

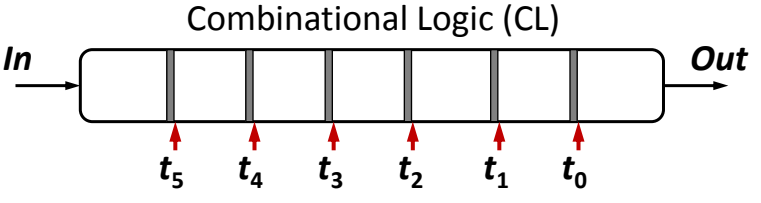
Timing requirement

$$T_{Cycle} > t_{Logic} + t_{Overhead}$$

Do we really need clocks?

Constant Propagation Delay?

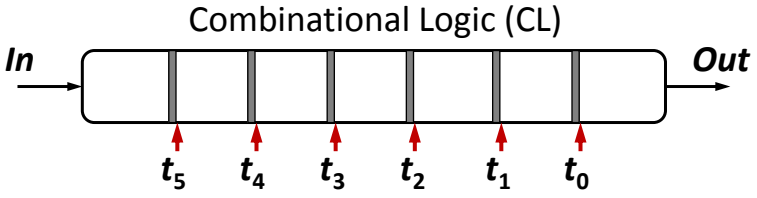
- **If the propagation delay of CL is constant (regardless of data input) and is known, we don't really need clocks**
  - It eliminates the $t_{Overhead}$
  - The inherent T_{Cycle} of a state-machine will be the delay
  - It can actually be even faster for data flow



$$T_{cycle} \approx \Delta t = t_1 - t_0$$

Wave Pipelining

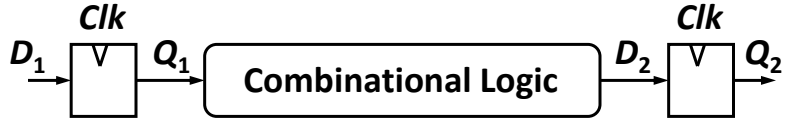
- **As the data is propagating down the logic chain ("wave"), a new "wave" can enter**
  - Provided the delay is constant
  - As we know, the delay is not constant

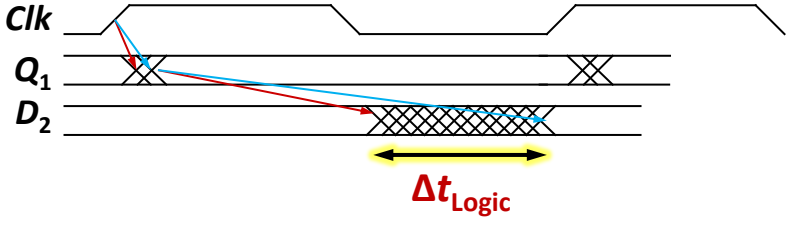


$$T_{cycle} \approx \Delta t = t_1 - t_0$$

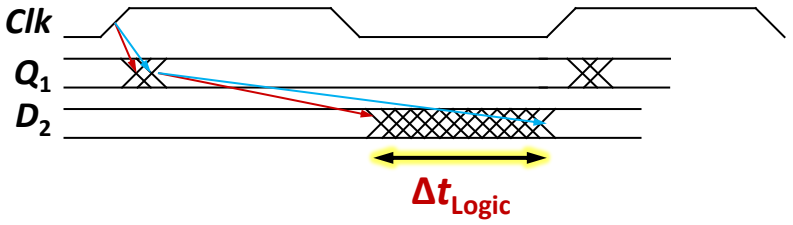
Variable Propagation Delay (1/2)

- **Delay through a combinational logic depends on**
  - Transition (rise/fall), type of logic, the input position…





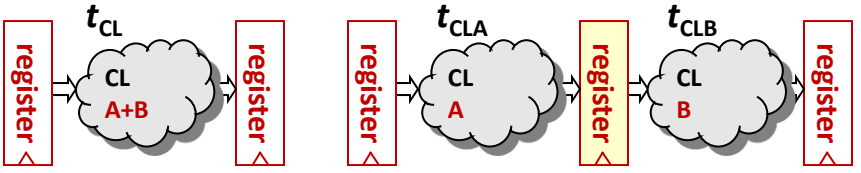
Variable Propagation Delay (2/2)



- For wave pipelining, the spread limits the "cycle time"
- Asynchronous uses the signals to indicate "completion," so the cycle time varies

Improving Throughput: Pipelining

- In a clocked system, just like wave pipelining but use clocks to remove the delay uncertainty
  - This allows 2 "waves" of data to be present inside CL



$T_{cycle} > t_{CL} + t_{Overhead}$ $T_{cycle} > \max(t_{CLA}, t_{CLB}) + t_{Overhead}$

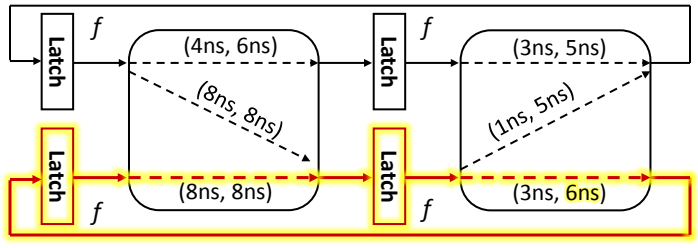
?

- Extend this ad infinitum?
  - Overhead eventually limits pipelining
  - Logic granularity limits the resolution

Examples

Example 10.1: 1-Φ Clock, Min T_{Cycle} = ?

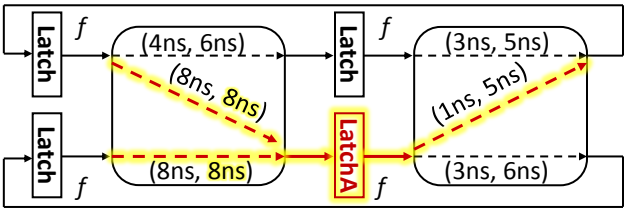
- **Assume:** $t_{D\text{-}Q} = t_{Setup} + t_{Clk\text{-}Q}$ = 0.5ns, t_{Hold} = 0.2ns
- The delays indicate (min, max) for different paths



$T_{cycle} \geq 8n + 6n + 2*0.5n = 15n$ (over 2 cycles)

$T_{cycle} \geq 7.5ns$

Example 10.2a: 1-Φ Clock, Min *W* = ?



Max delay

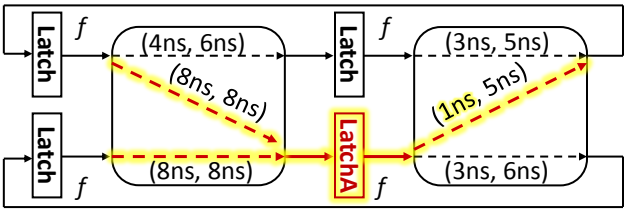
$$T_{Clk} + W > t_{Logic,max} + t_{D\text{-}Q}$$

from 10.20

W_{min} must handle time borrowing

- W > 8.0n (max delay) + 0.5n – 7.5n = 1n

Example 10.2b: 1-Φ Clock, Max *W* = ?



Min delay

$$t_{Hold} + W < t_B + t_{Clk\text{-}Q,cd} + t_{Logic,cd}$$

from 10.22

W_{max} must satisfy hold time

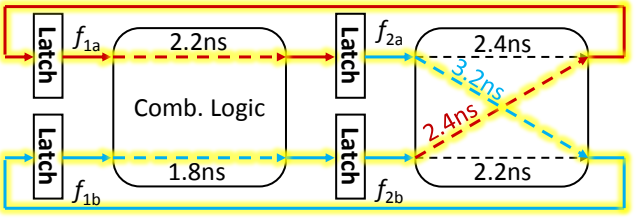
- *W* < 1n (min delay) + 0.5n – 0.2n = 1.3n

Note: $Latch_A$ is always borrowing t_B = 1ns!

- *W* < 1n + 1n (min delay) + 0.5n – 0.2 = 2.3n

Example 10.3: 2-Φ Clock, Min T_{Cycle} = ?

- **Assume:** $t_{D\text{-}Q} = t_{Setup} + t_{Clk\text{-}Q} = 0.2$ns



- **Top loop = 2.2 + 2.4 + 0.2*2 = 5ns**
- **Bottom loop = 1.8 + 2.2 + 0.2*2 = 4.4ns**
- **Cross loop (2 cycles) = 1.8 + 2.4 + 0.2*2 + 2.2 + 3.2 + 0.2*2 = 10.4ns**

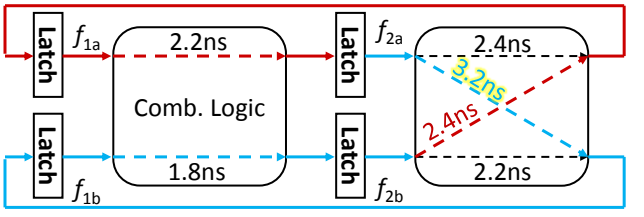
⇨ • **$T_{Cycle} \geq 5.2$ns**

(1st cycle done early to give time to 2nd cycle)

10.43

Example 10.4: 2-Φ Clock, Min W = ?

Assume: $t_{D\text{-}Q} = 0.2$ns, $T_{Cycle} = 5.2$ns, Pos Clk edge by $T_{Cycle}/2$



Max delay

$$T_{Clk}/2 + W > t_{Logic,max} + t_{D\text{-}Q}$$

from 10.27

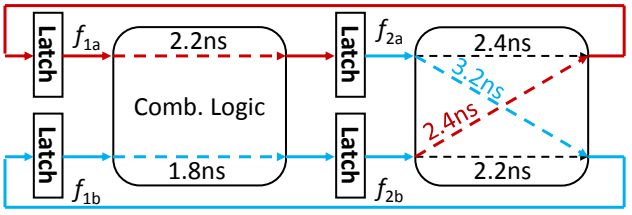
⇨ **Latch f_{1b} must latch in the data from f_{2a}**

- **$W > 3.2$ns + 0.2ns – 2.6ns = 0.8ns**

10.44

Example 10.5: 2-Φ Clock, Max t_{CU} = ?

Assume: $t_{D\text{-}Q}$ = 0.2ns, T_{Cycle} = 5.2ns, Pos Clk edge by $T_{Cycle}/2$



- ***W* > 0.8ns**

If $W = T_{Cycle}/4$ = 1.3ns, what's the max t_{CU}?

- **t_{CU} < 1.3ns – 0.8ns = 0.5ns**

10.45

Timing: Summary

- **Most systems today are synchronous**
  - Many systems now have some degree of asynchrony through using multiple phases or self-timing
- **Clocking is critical in guaranteeing functionality of a synchronous system to meet performance**
  - Delay can be too long so that data is not latched
  - Delay can be too short to cause data to race through
  - Clock has skew and can cause errors in timing

10.46

Clocking Methodologies: Summary

- **Three different clocking methodologies**
  - 2-phase, edge-triggered, pulse-mode
- **Each has their criteria on pulse width (duty cycle) and cycle time**
  - By using skew or pulse width appropriately, we can allow delays to exceed the cycle time through time borrowing